CS498 AMO Name: Xin Qu, ID: xinq2
Part 1 Accuracies

Setup	Cross_validation Accuracy
Unprocessed data	75.45%
0-value elements ignored	75.83%

1&2. Calculation of distribution parameters and Naive Bayes predictions

In [4]:
```
def norm_prob(x, mean_y, var_y): ### var = stdv**2
    ###return P(x|y)
    tmp = np.exp(-(x - mean_y) ** 2 / (2 * var_y))
    return 1 / ((2 * np.pi * var_y) ** 0.5) * tmp
```

In [5]:
```
def class_prob(xi, data):
    ###return class probability
    prob = {0: 1, 1: 1}
    for i in range(len(xi) - 1): ### xi needs to drop column('class')
        ###P(xi|y) = P(x0|y) * P(x1|y)*...
        prob[0] = prob[0] * norm_prob(xi[i], data['mean_y'].iloc[0][i], data['var_y'].iloc[0][i])
        prob[1] = prob[1] * norm_prob(xi[i], data['mean_y'].iloc[1][i], data['var_y'].iloc[1][i])
    prob[0] *= data['pt']
    prob[1] *= data['pn']
    return prob
def classify(xi, data):
    prob = class_prob(xi, data)
    return 0 if prob[0] > prob[1] else 1
```

In [6]:
```
def NBclassifier(dataset, ratio):
    accu_list = []
    for i in range(10):
        train, test = train_test_split(dataset, ratio)
        class_1 = train['class'][train['class'] == 1].count()
        class_0 = train['class'][train['class'] == 0].count()
        class_total = train.shape[0]
        pn = class_1 / float(class_total) ##1:negtaive
        pt = class_0 / float(class_total) ## 0: positive
        mean_y = train.groupby('class').mean()
        var_y = train.groupby('class').var()
        ##std_y = train.groupby('class').std()
        cor = 0
        for i, j in test.iterrows():
            result = classify(j, {'mean_y': mean_y, 'var_y': var_y, 'pn':pn, 'pt':pt})
            if bool(result) == bool(j['class']):
                cor += 1
        accu_list.append(cor / float(test.shape[0]))
    accuracy = np.sum(accu_list) / 10.0
    return accuracy
```

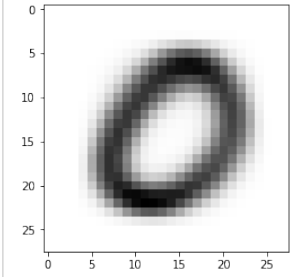
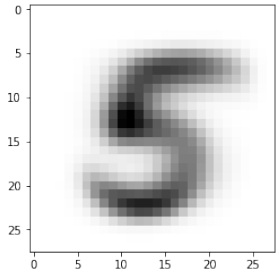
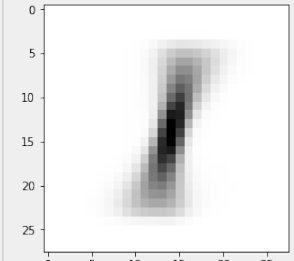
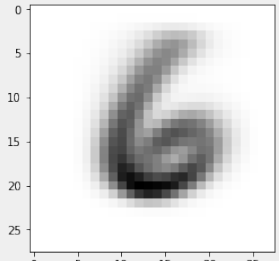
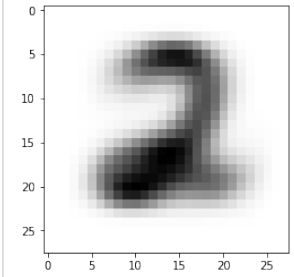
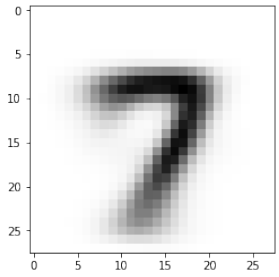
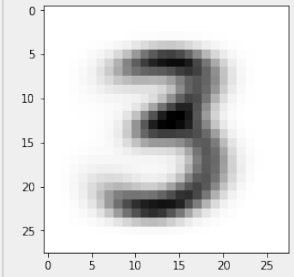
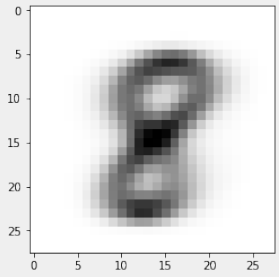
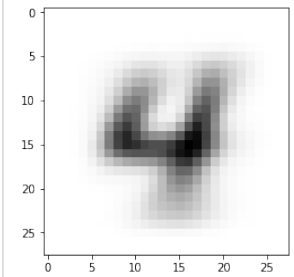
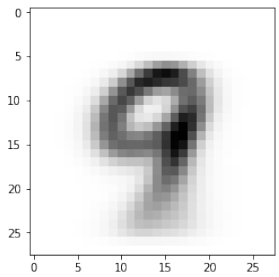
3. Test-train split code

In [3]:
```
import random
def train_test_split(dataset, ratio):
    data = dataset.values
    random.shuffle(data)
    size = len(data)
    train_size = int(size * (1 - ratio))
    train_data = data[:train_size, :]
    test_data = data[train_size:, :]
    train = pd.DataFrame(train_data, index = None,
                         columns = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'])
    test = pd.DataFrame(test_data, index = None,
                        columns = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'])
    train[['preg', 'plas', 'pres', 'skin', 'test', 'age', 'class']] =  train[['preg',
                                            'plas', 'pres', 'skin', 'test', 'age', 'class']].astype(int)
    test[['preg', 'plas', 'pres', 'skin', 'test', 'age', 'class']] = test[['preg',
                                            'plas', 'pres', 'skin', 'test', 'age', 'class']].astype(int)
    return train, test
```

Part 2 MNIST Accuracies

x	Method	Training Set Accuracy	Test Set Accuracy
1	Gaussian + untouched	77.75%	79.58%
2	Gaussian + stretched	79.75%	82.06%
3	Bernoulli + untouched	83.77%	85.17%
4	Bernoulli + stretched	82.50%	84.15%
5	10 trees + 4 depth + untouched	74.33%	76.15%
6	10 trees + 4 depth + stretched	71.62%	72.98%
7	10 trees + 16 depth + untouched	99.46%	94.89%
8	10 trees + 16 depth + stretched	99.48%	95.53%
9	30 trees + 4 depth + untouched	79.83%	80.61%
10	30 trees + 4 depth + stretched	74.20%	75.76%
11	30 trees + 16 depth + untouched	99.76%	96.20%
12	30 trees + 16 depth + stretched	99.72%	96.91%

Part 2A Digit Images

Digit	Mean Image	Digit	Mean Image
0		5	
1		6	
2		7	
3		8	
4		9	

1& 2. Calculation of Normal distribution and Bernoulli distribution parameters & 3. Calculation of Naive Bayes predictions

```
def Gaussiannb_predict(trainset, testset):
    def norm_prob_cal(x, mean_y, var_y): ### var = stdv**2
        ###return P(x|y)
        tmp = np.exp(-(x - mean_y) ** 2 / (2 * var_y))
        return 1 / ((2 * np.pi * var_y) ** 0.5) * tmp
    #model_nb = GaussianNB()#model_nb.fit(trainset, train_y)
    #mean = model_nb.theta_  ### mean values (10, 784)
    #var = model_nb.sigma_   ### var values  (10, 784)
    #pro = model_nb.class_prior_ ###prob of each class (10)
    from collections import Counter
    count_dic = Counter(testset)
    prob = []
    for i in count_dic.keys():
        p = count_dic[i] / float(sum(count_dic.values()))
        prob.append(p)
    prob = np.array(prob)
    columns = [str(i) for i in range(trainset.shape[1])]
    df = pd.DataFrame(trainset, index = None, columns = columns)
    df['class'] = testset
    mean_ = df.groupby('class').mean()
    var_ = df.groupby('class').var()
    pro = prob
    mean = mean_
    var = var_
    pred_y = []
    ###return predict of trainset
    for i in range(len(trainset)):
        norm_prob = norm_prob_cal(trainset[i], mean, var)
        norm_prob = norm_prob.replace(np.NaN, 1)
        norm_log_prob = np.log(norm_prob)
        p = np.sum(norm_log_prob, axis = 1) + np.log(pro)
        pred_y.append(np.argmax(p))
    return pred_y
```

```
def Bernoulli_pred(trainset_x, trainset_y):
    #model = BernoulliNB()
    #model.fit(trainset, train_y)
    #log_p_class = model.class_log_prior_
    #log_p_feature = model.feature_log_prob_
    columns = [str(i) for i in range(trainset_x.shape[1])]
    df = pd.DataFrame(trainset_x, index = None, columns = columns)
    df['class'] = trainset_y
    count_dic = Counter(trainset_y)
    prob = []
    for i in count_dic.keys():
        p = count_dic[i] / float(sum(count_dic.values()))
        prob.append(p)
    prob = np.array(prob)
    log_p_class = np.log(prob)
    class_dic = Counter(trainset_y)
    class_count = np.array(count_dic.values())
    feature_count = np.array(df.groupby('class').sum())
    log_p_feature = np.log(feature_count + 1.0) - np.log((class_count + 2.0).reshape(-1, 1))
    lpg_p_feature = np.array(log_p_feature)
    pred = []
    for i in range(len(trainset_x)):
        tmp = trainset_x[i] * log_p_feature + (1 - trainset_x[i]) * np.log((1 - np.exp(log_p_feature)))
        total_tmp = np.sum(tmp, axis = 1)
        p = total_tmp + log_p_class
        pred.append(np.argmax(p))
    return pred
```

4. Training of a decision tree & 5. Calculation of a decision tree predictions

```
resultB = defaultdict(list)
for (train, test, name) in dataset:
    for [tree, depth] in combinations:
        model = RandomForestClassifier(n_estimators = tree, max_depth = depth)
        model.fit(train, train_y)
        pred_y = model.predict(test)
        train_score = accuracy_score(map(int, train_y), model.predict(train))
        test_score = accuracy_score(map(int, test_y), pred_y)
        resultB[name].append((train_score, test_score))
```

Code for part 1

In [1]:
```
import pandas as pd
import numpy as np
df = pd.read_csv('pima-indians-diabetes.csv', header = None, names =
                 ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'])
df.head()
```

Out[1]:

	preg	plas	pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [2]:
```
df.shape
```

Out[2]: (768, 9)

In [3]:
```
import random
def train_test_split(dataset, ratio):
    data = dataset.values
    random.shuffle(data)
    size = len(data)
    train_size = int(size * (1 - ratio))
    train_data = data[:train_size, :]
    test_data = data[train_size:, :]
    train = pd.DataFrame(train_data, index = None,
                         columns = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'])
    test = pd.DataFrame(test_data, index = None,
                        columns = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'])
    train[['preg', 'plas', 'pres', 'skin', 'test', 'age', 'class']] =  train[['preg',
                                            'plas', 'pres', 'skin', 'test', 'age', 'class']].astype(int)
    test[['preg', 'plas', 'pres', 'skin', 'test', 'age', 'class']] = test[['preg',
                                            'plas', 'pres', 'skin', 'test', 'age', 'class']].astype(int)
    return train, test
```

In [4]:
```
def norm_prob(x, mean_y, var_y): ### var = stdv**2
    ###return P(x|y)
    tmp = np.exp(-(x - mean_y) ** 2 / (2 * var_y))
    return 1 / ((2 * np.pi * var_y) ** 0.5) * tmp
```

In [5]:
```
def class_prob(xi, data):
    ###return class probability
    prob = {0: 1, 1: 1}
    for i in range(len(xi) - 1): ### xi needs to drop column('class')
        ###P(xi|y) = P(x0|y) * P(x1|y)*...
        prob[0] = prob[0] * norm_prob(xi[i], data['mean_y'].iloc[0][i], data['var_y'].iloc[0][i])
        prob[1] = prob[1] * norm_prob(xi[i], data['mean_y'].iloc[1][i], data['var_y'].iloc[1][i])
    prob[0] *= data['pt']
    prob[1] *= data['pn']
    return prob
def classify(xi, data):
    prob = class_prob(xi, data)
    return 0 if prob[0] > prob[1] else 1
```

In [6]:
```
def NBclassifier(dataset, ratio):
    accu_list = []
    for i in range(10):
        train, test = train_test_split(dataset, ratio)
        class_1 = train['class'][train['class'] == 1].count()
        class_0 = train['class'][train['class'] == 0].count()
        class_total = train.shape[0]
        pn = class_1 / float(class_total) ##1:negtaive
        pt = class_0 / float(class_total) ## 0: positive
        mean_y = train.groupby('class').mean()
        var_y = train.groupby('class').var()
        ##std_y = train.groupby('class').std()
        cor = 0
        for i, j in test.iterrows():
            result = classify(j, {'mean_y': mean_y, 'var_y': var_y, 'pn':pn, 'pt':pt})
            if bool(result) == bool(j['class']):
                cor += 1
        accu_list.append(cor / float(test.shape[0]))
    accuracy = np.sum(accu_list) / 10.0
    return accuracy
```

In [7]:
```
accuracy = NBclassifier(df, 0.2)
print('The estimate of accuracy of the classifier is %.2f' %(accuracy * 100) + '%')
```

```
The estimate of accuracy of the classifier is 75.45%
```

part 1B

In [8]:
```
df_B = pd.read_csv('pima-indians-diabetes.csv', header = None,
                   names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'])
df_B[['pres', 'skin', 'mass', 'age']] = df_B[['pres', 'skin', 'mass', 'age']].replace(0, np.NaN)
df_B = df_B.dropna()
df_B.head()
df_B.shape
```

Out[8]: (537, 9)

In [9]:
```
accuracy_B = NBclassifier(df_B, 0.2)
```

In [10]:
```
print('The estimate of accuracy of the modified classifier by is %.2f' %(accuracy_B * 100) + '%')
```

```
The estimate of accuracy of the modified classifier by is 75.83%
```

Code for part 2

```
from mnist import MNIST
mndata = MNIST('/Users/xinqu/Sandbox/CS498 Applied Machine Learning/HW/HW1/python-mnist/data')
train_x, train_y = mndata.load_training()
test_x, test_y = mndata.load_testing()
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

```
from skimage.transform import resize
def crop_image(x):
    hor_max, ver_max = np.max(np.where(x != 0), 1)
    hor_min, ver_min = np.min(np.where(x != 0), 1)
    bound = x[hor_min: hor_max, ver_min: ver_max]
    resize_x = resize(bound, (20, 20), mode = 'constant')
    return resize_x
```

```
def threshold(x):
    y = x.copy()
    grey_y = y[y > 0]  ## remove 0 pixel
    mid_grey = sum(grey_y) / len(grey_y)
    y[y < mid_grey] = 0
    y[y > 0] = 1
    return y
```

```
import numpy as np
train_x = np.array(train_x).astype(np.uint8)
train_y = np.array(train_y).astype(np.uint8)
test_x = np.array(test_x).astype(np.uint8)
test_y = np.array(test_y).astype(np.uint8)
```

```
##untouched image
###threshold pixel
train_x_thre = []
for x in train_x:
    train_x_thre.append(x)
train_x_thre = np.array([threshold(x) for x in train_x_thre])
test_x_thre = []
for x in test_x:
    test_x_thre.append(x)
test_x_thre = np.array([threshold(x) for x in test_x_thre])
```

```
##stretched image
###crop image
train_x_crop = []
test_x_crop = []
for x in train_x.reshape(-1, 28, 28):
    train_x_crop.append(np.ravel(crop_image(x)))
for x in test_x.reshape(-1, 28, 28):
    test_x_crop.append(np.ravel(crop_image(x)))
test_x_crop = np.array(test_x_crop)
###
train_x_crop_thre = np.array([threshold(x) for x in train_x_crop])
test_x_crop_thre = np.array([threshold(x) for x in test_x_crop])
```

```
def accuracyscore(x, y):
    return sum(x == y) / float(len(y))
```

```python
def Gaussiannb_predict(trainset, testset):
    def norm_prob_cal(x, mean_y, var_y): ### var = stdv**2
        ###return P(x|y)
        tmp = np.exp(-(x - mean_y) ** 2 / (2 * var_y))
        return 1 / ((2 * np.pi * var_y) ** 0.5) * tmp
    #model_nb = GaussianNB()#model_nb.fit(trainset, train_y)
    #mean = model_nb.theta_  ### mean values (10, 784)
    #var = model_nb.sigma_   ### var values  (10, 784)
    #pro = model_nb.class_prior_ ###prob of each class (10)
    from collections import Counter
    count_dic = Counter(testset)
    prob = []
    for i in count_dic.keys():
        p = count_dic[i] / float(sum(count_dic.values()))
        prob.append(p)
    prob = np.array(prob)
    columns = [str(i) for i in range(trainset.shape[1])]
    df = pd.DataFrame(trainset, index = None, columns = columns)
    df['class'] = testset
    mean_ = df.groupby('class').mean()
    var_ = df.groupby('class').var()
    pro = prob
    mean = mean_
    var = var_
    pred_y = []
    ###return predict of trainset
    for i in range(len(trainset)):
        norm_prob = norm_prob_cal(trainset[i], mean, var)
        norm_prob = norm_prob.replace(np.NaN, 1)
        norm_log_prob = np.log(norm_prob)
        p = np.sum(norm_log_prob, axis = 1) + np.log(pro)
        pred_y.append(np.argmax(p))
    return pred_y
```

```python
accuracyscore(Gaussiannb_predict(train_x_thre, train_y), train_y) ###Gaussian train set untouched
```

```
/anaconda2/lib/python2.7/site-packages/ipykernel_launcher.py:30: RuntimeWarning: divide by zero encountered in log
```

```
0.7775333333333333
```

```python
accuracyscore(Gaussiannb_predict(train_x_crop_thre, train_y), train_y) ###Gaussian train set stretched
```

```
/anaconda2/lib/python2.7/site-packages/ipykernel_launcher.py:30: RuntimeWarning: divide by zero encountered in log
```

```
0.79585
```

```python
accuracyscore(Gaussiannb_predict(test_x_thre, test_y), test_y) ###Gaussian test set untouched
```

```
0.7975
```

```python
accuracyscore(Gaussiannb_predict(test_x_crop_thre, test_y), test_y) ###Gaussian test set stretched
```

```
0.8206
```

```python
def Bernoulli_pred(trainset_x, trainset_y):
    #model = BernoulliNB()
    #model.fit(trainset, train_y)
    #log_p_class = model.class_log_prior_
    #log_p_feature = model.feature_log_prob_
    columns = [str(i) for i in range(trainset_x.shape[1])]
    df = pd.DataFrame(trainset_x, index = None, columns = columns)
    df['class'] = trainset_y
    count_dic = Counter(trainset_y)
    prob = []
    for i in count_dic.keys():
        p = count_dic[i] / float(sum(count_dic.values()))
        prob.append(p)
    prob = np.array(prob)
    log_p_class = np.log(prob)
    class_dic = Counter(trainset_y)
    class_count = np.array(count_dic.values())
    feature_count = np.array(df.groupby('class').sum())
    log_p_feature = np.log(feature_count + 1.0) - np.log((class_count + 2.0).reshape(-1, 1))
    lpg_p_feature = np.array(log_p_feature)
    pred = []
    for i in range(len(trainset_x)):
        tmp = trainset_x[i] * log_p_feature + (1 - trainset_x[i]) * np.log((1 - np.exp(log_p_feature)))
        total_tmp = np.sum(tmp, axis = 1)
        p = total_tmp + log_p_class
        pred.append(np.argmax(p))
    return pred
```

```python
accuracyscore(Bernoulli_pred(train_x_thre, train_y), train_y)## Bernoulli untouched train set
```

```
0.8376833333333333
```

```python
accuracyscore(Bernoulli_pred(test_x_thre, test_y), test_y)## Bernoulli untouched test set
```

```
0.8517
```

```
accuracyscore(Bernoulli_pred(train_x_crop_thre, train_y), train_y)## Bernoulli stretched train set
```

```
0.8250333333333333
```

```
accuracyscore(Bernoulli_pred(test_x_crop_thre, test_y), test_y)## Bernoulli stretched test set
```

```
0.8415
```

plot mean pixel values for Normal Distribution

```
columns = [str(i) for i in range(train_x_thre.shape[1])]
df = pd.DataFrame(train_x_thre, index = None, columns = columns)
df['class'] = train_y
mean_ = np.array(df.groupby('class').mean())
for i in range(len(mean_)):
    image = mean_[i]
    plt.imshow(image.reshape((28, 28)), cmap = 'Greys')
    plt.show()
```

```
###compare with library build-in predict function
dataset = [(train_x_thre, test_x_thre, 'untouched'), (train_x_crop_thre, test_x_crop_thre, 'stretched')]
from collections import defaultdict
result = defaultdict(list)
for (train, test, name) in dataset:
    for model in [GaussianNB(), BernoulliNB()]:
        model.fit(train, train_y)
        pred_y = model.predict(test)
        train_score = accuracy_score(map(int, train_y), model.predict(train))
        test_score = accuracy_score(map(int, test_y), pred_y)
        result[name].append((train_score, test_score))
print(result)
### result {'untouched': [(Gaussian_train_accu, Gaussian_test_accu),
###                       (Bernoulli_train_accu, Bernoulli_test_accu)],
###         'stretched': [(Gaussian_train_accu, Gaussian_test_accu),
###                       (Bernoulli_train_accu, Bernoulli_test_accu)]}
```

```
defaultdict(<type 'list'>, {'untouched': [(0.54855, 0.5399), (0.8376833333333333, 0.8487)], 'stretched': [(0.79525,
0.8075), (0.8250333333333333, 0.8381)]})
```

Part 2B: MNIST using Decision Forest

```
from sklearn.ensemble import RandomForestClassifier
```

```
import itertools
trees = [10, 30]
depth = [4, 16]
combinations = list(itertools.product(trees, depth))
combinations
```

```
[(10, 4), (10, 16), (30, 4), (30, 16)]
```

```
resultB = defaultdict(list)
for (train, test, name) in dataset:
    for [tree, depth] in combinations:
        model = RandomForestClassifier(n_estimators = tree, max_depth = depth)
        model.fit(train, train_y)
        pred_y = model.predict(test)
        train_score = accuracy_score(map(int, train_y), model.predict(train))
        test_score = accuracy_score(map(int, test_y), pred_y)
        resultB[name].append((train_score, test_score))
print(resultB)
###resultB = {'untouched': [(train_accu 10trees + 4depth, test_accu 10trees + 4depth),
###                         (train_accu 10trees + 16depth, test_accu 10trees + 16depth),
###                         (train_accu 30trees + 4depth, test_accu 30trees + 4depth),
###                         (train_accu 30trees + 16depth, test_accu 30trees + 4depth)],
###           'stretched': [(train_accu 10trees + 4depth, test_accu 10trees + 4depth),
###                         (train_accu 10trees + 16depth, test_accu 10trees + 16depth),
###                         (train_accu 30trees + 4depth, test_accu 30trees + 4depth),
###                         (train_accu 30trees + 16depth, test_accu 30trees + 4depth)}
```

```
defaultdict(<type 'list'>, {'untouched': [(0.7433166666666666, 0.7615), (0.9946333333333334, 0.9489), (0.7983, 0.806
1), (0.9976, 0.962)], 'stretched': [(0.7162166666666666, 0.7298), (0.9948333333333333, 0.9553), (0.742, 0.7576), (0.9
970833333333333, 0.9672)]})
```